Reinforcement Learning 2

Complications & approximations

Andrew Lampinen

Psych 209, Winter 2018

Introduction

Plan for this lecture

Talk about all the stuff that makes it messy:

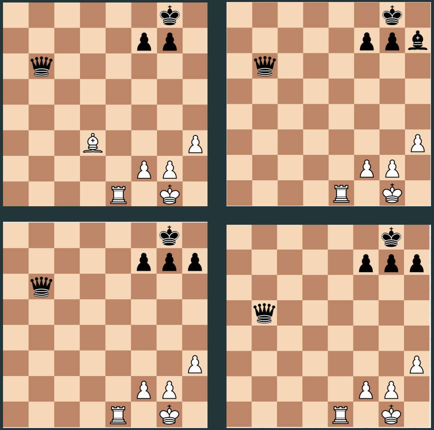


- Infinite spaces, approximation & generalization.
- Correlations & replay.
- Exploration & on/off-policy learning.
- Hierarchies & plans.
- Other types of feedback.
- Unobservables & POMDPs, different assumptions and other approaches.

Function approximation

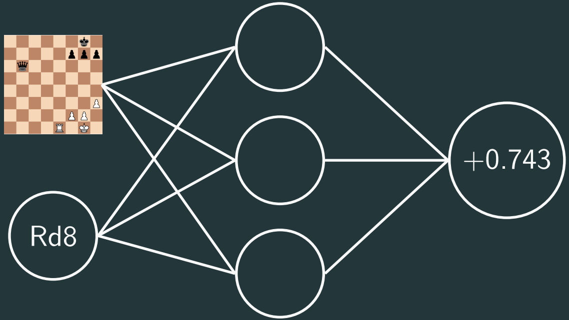
Similarities & dissimilarities

- Wheels of car can be at infinitely many angles, but is 0.55 radians really that different from 0.56?
- More state, action pairs in chess than atoms in observable universe. However:



Approximating the Q-table

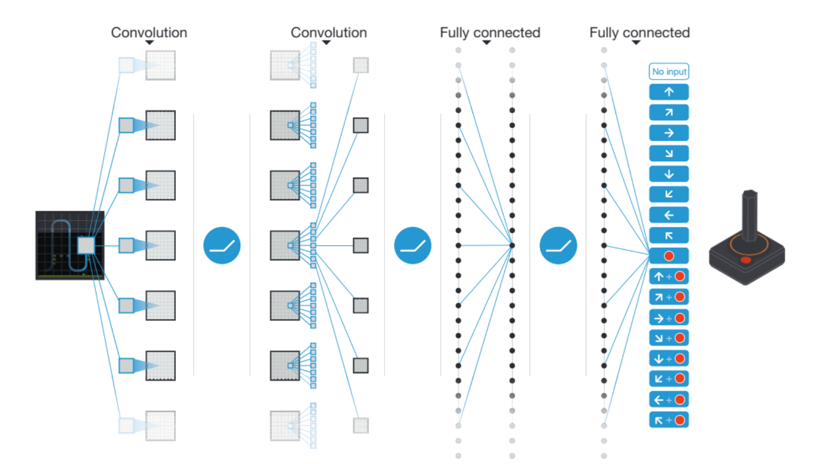
- Previously, the Q-table was a lookup function. Let's replace this with some other function that maps states and actions to Q-values.



- Loss:

$$L = \left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_i^-) - Q(s_t, a_t; \theta_i) \right)^2$$

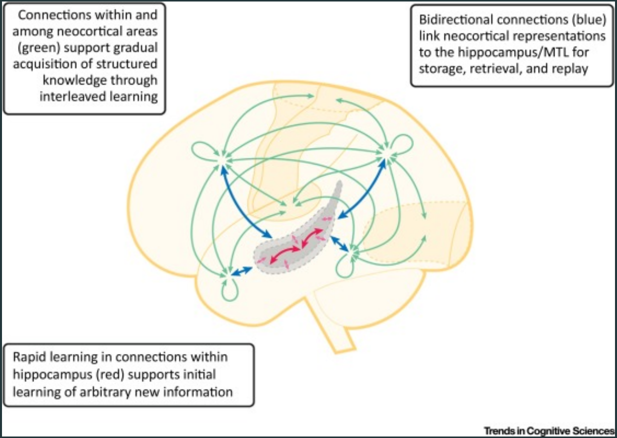
Playing atari games



Replay

Fix one problem and ...

- Generalization comes at the expense of cross talk and catastrophic forgetting. Will a self-driving car learning to parallel park forget how to drive on the freeway?
- CLS is a theory of how humans and animals avoid this:



Replay buffers

The Atari game-playing paper addressed this with **experience replay**:

- Whenever we experience a new $(s_t, a_t, r_{t+1}, s_{t+1})$ tuple, stick it in a buffer.
- instead of

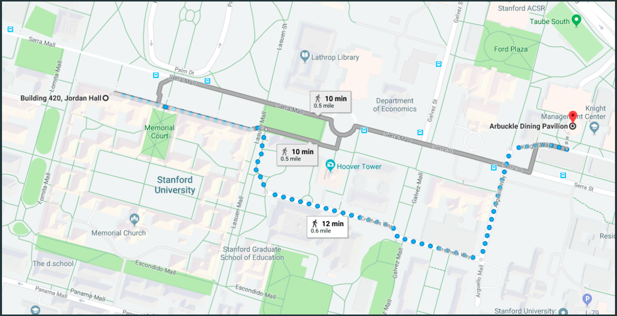
$$\Delta Q^{\pi}(s_t, a_t) = \alpha \underbrace{\left(\left[r_{t+1} + \max_{a'} \gamma Q^{\pi}(s_{t+1}, a') \right] - Q(s_t, a_t) \right)}_{\text{prediction error!}}$$

- we sample a random experience from our buffer and update with that: $k \sim \mathit{Unif}(\text{replay buffer indices})$

$$\Delta Q^{\pi}(s_k, a_k) = \alpha \underbrace{\left(\left[r_{k+1} + \max_{a'} \gamma Q^{\pi}(s_{k+1}, a') \right] - Q(s_k, a_k) \right)}_{\text{prediction error!}}$$

Exploration and exploitation and on/off policy

The need to explore



- How do I trade off between **exploiting** the best path to lunch I've found so far and **exploring** my other options?
- In other words, how do we incorporate exploration into our policy?
- This is important – remember convergence guarantees required *every* state, action pair to be visited "frequently."

The exploration-exploitation trade-off

There is a *fundamental* trade-off between exploring and exploiting:

- Exploring wastes time trying things I'm pretty sure aren't good.
- Exploiting risks missing a great opportunity.
- We have to find some balance between these that results in good payoffs but still makes sure we don't miss too much.

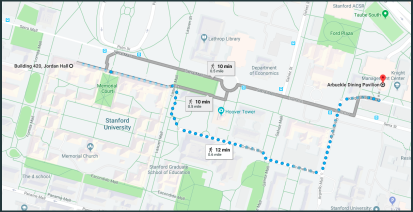
ϵ-greedy

- One simple technique is just to choose actions randomly some small fraction ϵ of the time.
- The rest of the time, take the maximum Q action.
- We call this ϵ-greedy.
- We can also do clever things like *anneal* ϵ over training.
    - Act mostly randomly and explore a lot early in training.
    - Act mostly greedily and exploit a lot late in training/in testing.
- (There are other possibilities, e.g. using a softmax over Q values.)

Exploit during testing or keep exploring?

The point about testing vs. training is a little tricky...

- Do chess players stop learning when they're playing for the world championship?
- A self-driving car can't just be trained and set free – destinations, roads, and laws are all evolving.
- What if my direct path to lunch was blocked by a building that was torn down?



Non-optimality of Q-learning under exploration

- But Q-learning fundamentally assumes that we will be behaving completely greedily during training! Why?
- It's built into our computation of the Q-update:

$$\Delta Q^{\pi}(s_t, a_t) = \alpha\left(\left[r_{t+1} + \max_{a'} \gamma Q^{\pi}(s_{t+1}, a')\right] - Q(s_t, a_t)\right)$$

- If we want to behave optimally with a policy π that explores, we have to incorporate this policy into the learning rule:

$$\Delta Q^{\pi}(s_t, a_t) \stackrel{?}{=} \alpha\left(\left[r_{t+1} + \sum_{a'} p(a'|s_t, a_t, s_{t+1}, \pi)\, \gamma Q^{\pi}(s_{t+1}, a')\right] - Q(s_t, a_t)\right)$$

SARSA

- One algorithm for doing this is SARSA. It works almost exactly like Q learning, except we store $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$
- and change our update from

$$\Delta Q^\pi(s_t, a_t) = \alpha\left(\left[r_{t+1} + \max_{a'} \gamma Q^\pi(s_{t+1}, a')\right] - Q(s_t, a_t)\right)$$

to

$$\Delta Q^\pi(s_t, a_t) = \alpha([r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1})] - Q(s_t, a_t))$$

- Since our actions in the state are distributed according to π, in expectation our Q-value updates will be as well.

On/off-policy

This highlights an important, but somewhat orthogonal distinction:

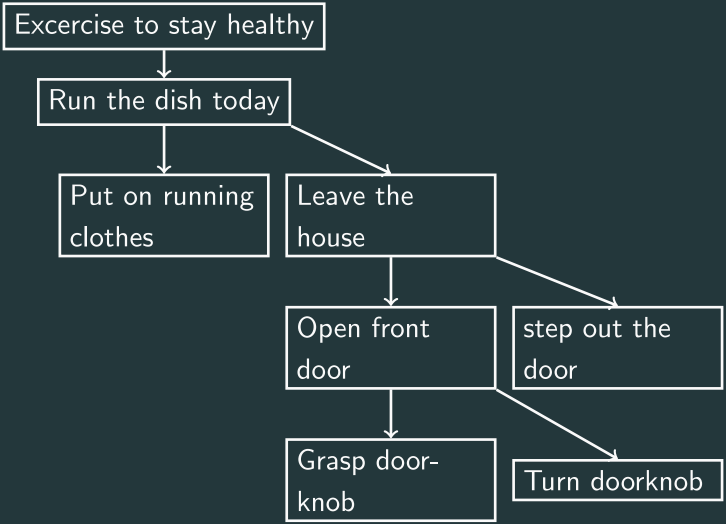
- Do we want to learn **on-policy**, that is, using the same policy we will be using during evaluation (like SARSA)?
- Or do we want to learn **off-policy**, that is, using a different policy during training than during evaluation (like Q-learning)?

There are tradeoffs!

- **On-policy:** Can be faster, can be more stable.
- **Off-policy:** Can learn from anything, even totally random play. This makes incorporating replay or changing ϵ for more early exploration easier.

Hierarchies & plans

Plans = hierarchies of actions



Hierarchies of Q-learning?

- You could think of doing Q-learning at each level, where state includes states of the above levels (including their goals and current higher-level actions, etc).
- But how do you figure out what the higher level actions should be?
- Still an open problem (though there is work on it), and potentially a good project is to think about how some aspect of behavior could be explained this way, and what assumptions you would need to make it work!

Other types of feedback

Auxiliary tasks

- Actually, we might get other forms of feedback... Can we incorporate these?
- Short answer: yes.
- May help with picking up on longer-term structure.
- Or with building relevant prior knowledge.

Other approaches

Other approaches

Many other approaches:

- Policy gradient methods.
- Actor-critic methods.
- Many, many variations.

And under different assumptions:

- What if we only observe part of the state? (POMDPs.)
- What if we actually build models of the world and plan over these? (Model-based RL, as opposed to Model-free.)
- ... Or something in between? (Successor-representation, imagination.)
- And much more.

Wrapping up

Summary

- We have an **agent** that takes **actions** in **states**.
- These actions are chosen according to a **policy**, sometimes derived from some sort of **value function**.
- We evaluated on **expected discounted rewards.**
- The **discount** reflects how future-oriented we are.
- $Q^\pi(s, a)$ is the expected return of a in s under π.
- We can learn Q values by **TD learning** (surprise).
- Can approximate Q using deep learning (for generalization).
- There's a trade-off between **exploring** and **exploiting**!
- ... and there's way more to RL than will fit on one slide or in one course.